Towards a Provenance-Aware Internet of Things (IoT) System

Ebelechukwu Nwafor *, Gedare Bloom*, Andre Campbell* and David Hill*
*Department of Electrical Engineering and Computer Science
Howard University, Washington, DC 30332–0250
Email: ebelechukwu.nwafor@bison.howard.edu

***Abstract*—The Internet of Things (IoT) offers immense benefits by enabling devices to leverage networked resources theryby making intelligent decisions. The numerous heterogeneous connected devices that exist throughout the IoT system creates new security and privacy concerns. Some of these concerns can be overcome through trust, transparency, and integrity, which can be achieved with data provenance. Data provenance, also known as data lineage, provides a history of transformations that occurs on a data object from the time it was created to its current state. Data provenance has been explored in the areas of scientific computing, business, forensic analysis, and intrusion detection. Data provenance can help in detecting and mitigating malicious cyber attacks. In this paper, we explore the integration of provenance within the IoT. We propose a provenance collection framework for IoT applications. We evaluate the effectiveness of our framework by looking at an application of provenance data by developing a prototype system for proof of concept.**

I. Introduction

The Internet of Things (IoT) has generated a lot of buzz among commercial and industrial information technology experts all over the world. Heterogeneous devices like we have never seen before are communicating with each other over a shared network (e.g internet). For example, It is possible to automatically control the temperature of a house remotely though a cell phone.

IoT offers a lot of benefits in the areas of home and industrial automation which makes operations which might require a lot of interaction seamless however, with this unprecedented communication, there has been an exponential increase in the number of devices connected to the internet. It is estimated that over 50 million devices will be connected to the internet by the year 2020. With the vast amounts of connected heterogeneous devices, security and privacy risks is increased. IoT devices are not strongly incorporated with security in mind. This raises the complexity of including security after the device has been deployed. For instance some devices come with default passwords which might never be changed during its lifecycle. The amount of data generated from IoT devices requires stronger levels of trust which can be achieved through data provenance. Rapid7 [**?**], an internet security and analytics organization, released a report highlighting vulnerabilities that exist on select IoT devices. In their report, they outline vulnerabilities in baby monitors which allowed intruders unauthorized access to devices whereby a malicious intruder can view live feeds from a remote location. With provenance information, we can generate an activity trail which can be further analyzed to determine who, where, and how, a malicious attack occurred in order to provide preventive measures to eradicate future or current attacks. [**?**].

Data provenance is a comprehensive history of activities that occurred on an entity from its origin to its current state. Provenance ensures integrity of data . Provenance has been applied in various area such as scientific workflow for experiment reproducibility, and information security as a form of access control and also for intrusion detection in mitigating malicious adversaries. Provenance ensure trust and integrity of data. IoT devices (things) produces sensor-actuator data. A workflow representation of of how sensor data is generated can be generated to depict dependency between sensor-actuator readings and devices/sensor information contained in the device.

This information generated can be prove to be beneficial as a means for mitigating malicious intrusion or for scientific reproducability as provenance. In this paper, we propose a provenance aware framework for IoT devices, in which provenance data is collected and modeled to represent dependencies between sensor-actuator readings and the various entities contained in the IoT architecture. Most of the interconnected heterogeneous devices (things) are embedded systems which require lightweight and efficient solutions as compared to general purpose systems. This requirement is attributed to the constrained memory and computing power of such devices.

The remaining section of the paper is organized as follows: section 2 discusses background information on IoT definition, architecture ,application domain and data provenance. Section 3 discusses the need for a incoporating provenance to IoT using a use case scenario of an automated smart home. Section 3 talks about related work in provenance collection systems. Section 4 discusses implementation details for provenance collection framework. Section 5 talks about results and experiment analysis. Finally, in section 6 we conclude with future work.

II. Background

This section describes key concepts of data provenance, IoT characteristics, and provenance models. It also provides motivating example for the need for provenance collection via a use case.

A. *Internet of Things*

There is no standard definition for IoT, however, researchers have tried to define the concept of connected things. The concept of IoT was proposed by Mark Weiser in the early 1990s which represents a way in which the physical objects, things, can be connected to the digital world. Gubbi et al defines the IoT as an interconnection of sensing and actuating devices that allows data sharing across platforms through a centralized framework. We define (IoT) as follows:

The Internet of Things (IoT) is a network of heterogeneous devices with sensing and actuating capabilities communicating over the internet.

IoT has applications in home automation, smart health, automotive communication, machine to machine comminication, industrial automation. The notion of IoT has been attributed to smart devices. The interconnectivity between various heterogeneous devices allows for devices to share information in a unique manner. Analytics is a driving force for IoT. With analytics, devices can learn from user data to make smarter decisions. This notion of smart devices is seen in various commercial applications such as smartwatches, thermostats that automatically learns a user patterns. The ubiquitous nature of these devices make them ideal choices to be included in consumer products. IoT architecture represents a functional hierarchy of how information is disseminated across multiple hierarchies contained in an IoT framework; from devices which contain sensing and actuating capabilities to massive data centers (cloud storage). Knowing how information is transmitted across layers allows a better understanding on how to model the flow of information across actors contained in an IoT hierarchy. Figure 1 displays the IoT architecture and the interactions between the respective layers. IoT architecture consists of four distinct layers: The sensor and actuator layer, device layer, gateway layer and the cloud layer. The base of the architectural stack consist of sensors and actuators which gathers provenance information and interacts with the device layer. The device layer consists of devices (e.g mobile phones, laptops, smart devices) which are responsible for aggregating data collected from sensors and actuators. These devices in turn forwards the aggregated data to the gateway layer. The gateway layer routes and forwards data collected from the device later. It could also serve as a medium of temporary storage and data processing. The cloud layer is involved with the storage and processing of data collected from the gateway layer. Note that the resource constraints decreases up the architectural stack with the cloud layer having the most resources (memory, power computation) and the sensor-actuator layer having the least.

With the recent data explosion [22] due to the large influx in amounts of interconnected devices, information is disseminated at a fast rate and with this increase involves security and privacy concerns. Creating a provenance-aware system is beneficial to IoT because it ensures the trust and integrity of interconnected devices. Enabling provenance collection in IoT devices allows these devices to capture valuable information

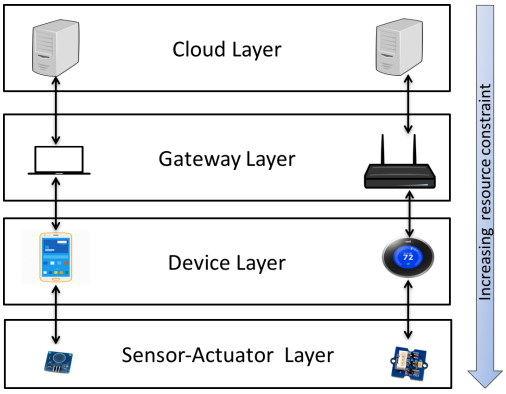

Fig. 1. IoT Architecture Diagram. The arrows illustrates the interaction between data at various layers on the architecture.

which enables backtracking in an event of a malicious attack.

B. *Provenance-Aware IoT Device Use Case*

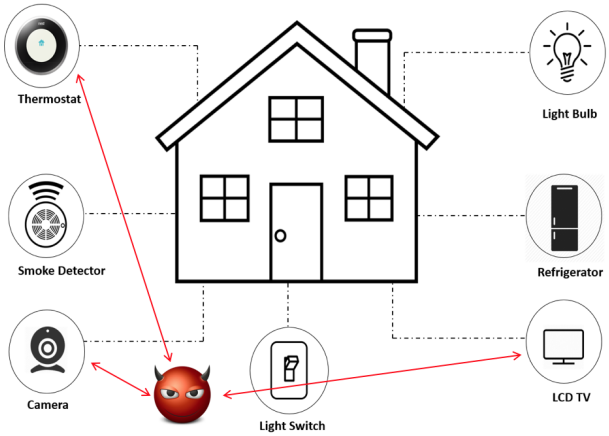

Fig. 2. Smart home use case Diagram

Consider a smart home as illustrated in Figure 2 that contains interconnected devices such as a thermostat which automatically detects and regulates the temperature of a room based on prior information of a user's temperature preferences, a smart lock system that can be controlled remotely and informs a user via short messaging when the door has been opened or is closed, a home security camera monitoring system, a smart fridge which sends a reminder when food products are low. In an event that a malicious intruder attempts to gain access to the smart lock system and security camera remotely, provenance information can be used to track the series of events to determine where and how a malicious attack originated. Provenance can also be used as a safeguard to alert of a possible remote or insider compromise thereby protecting against future or ongoing malicious attacks.

C. Data Provenance

The Oxford English dictionary defines provenance as the place of origin or earliest known history of something". An example of provenance can be seen with a college transcript. A transcript is the provenance of a college degree because it outlines all of the courses satisfied in order to attain the degree. In the field of computing, data provenance, also known as data lineage, can be defined as the history of all activities performed on entities from its creation to its current state. Cheney et al. describes provenance as the origin and history of data from its lifecycle. Buneman et al describes provenance from a database perspective as the origin of data and the steps in which it is derived in the database system. We formally define provenance as follows: Data provenance of an entity is a comprehensive history of activities that occur on that entity from its creation to its present state.

Provenance ensures trust and integrity of data [**?**]. It outlines dependency between all objects involved in the system and allows for the verification of the source of data. dependency is used to determine the relationship between multiple objects. The relationship in which provenance denotes can in turn be used in digital forensics [**?**] to investigate the cause of a malicious attack and also in intrusion detection systems to further enhance the security of computing devices. Provenance has been utilized in application domains such as computer security for access control and intrusion detection, in scientific experiments for reproducability and in version control systems to mention but a few.

1) Provenance Characteristics: Since provenance denotes the who, where and why of data transformation, it is imperative that data disseminated in an IoT architecture satisfies the required conditions. The characteristics of data provenance are outlined in detail below.

- Who: This characteristic provides information on activities made to an entity. Knowing the "who" characteristic is essential because it maps the identity of modification to a particular data object. An example of "who" in an IoT use case is a sensor device identifier.
- Where: This characteristic denotes location information in which data transformation was made. This provenance characteristic could be optional since not every data modification contains location details.
- When: This characteristic denotes the time information at which data transformation occurred. This is an essential provenance characteristic. Being able to tell the time of a data transformation allows for tracing data irregularities.
- What: This characteristic denotes the transformation is applied on a data object. A use case for IoT can be seen in the various operations (create, read, update, and delete) that could be performed on a file object.

2) Differentiating Provenance, Log data, and Metadata : Provenance has often been seen used interchangably to define log data, and metadata. While some overlap exists between the three, some differences exists. Log data contains information about the activities of an operating system or processes. Log data can be used as provenance because It contains data trace specific to an application domain. Log files might contain unrelated information such as error messages, warnings which might not be considered as provenance data. Provenance allows for specified collection of information that relates to the change of the internal state of an object. In summary, log data could provide insight to what provenance data to collect.

D. Model for representing provenance for IoT

In order to represent the right kind of provenance information in an IoT architecture, we need to satisfy the who, where, how, and what of data transformations. The provenance of sensor and actuator data in an IoT device showing the depency between all entities responsible for generating the sensor reading.

Provenance data can be represented using a provenance model in a modeling language such as, PROVDM which is represented in serialized in three format: XML, JSON and RDF. This model displays data dependency between data objects. We propose a model that contains information such as sensor readings, device name, and device information. There are two widely accepted modeling languages for representing provenance, PROV-DM [**?**] and Open Provenance Model [**?**] that have been applied in various literature and are considered standard for representing provenance. Details on the provenance models are outlined below.

1) Provenance Data Model (Prov-DM): PROV-DM is a model that conforms to PROV ontology. It is a W3C standard that is used to depict dependencies between entities, activities and agents (digital or physical). It creates a common model that allows for interchange of provenance information between heterogeneous devices. It contains two major components: types and relations.

- Entity: An entity is a physical or digital object. An example of an entity is a file system, a process, or an motor vehicle. An entity may be physical or abstract.
- Activity: An activity represents some form of action that occurs over a time frame. Actions are acted upon by an agent. An example of an activity is a process opening a file directory, Accessing a remote server.
- Agent: An agent is a thing that takes ownership of an entity, or performs an activity. An example of an agent is a person, a software product, or a process.

The figure below illustrates the various types contained in PROV-DM and their representation. Entities, activities and agents are represented by oval, rectangle and hexagonal shapes respectively.

PROV-DM does not keep track of future events. PROV-DM relations are outlined below:

- wasGeneratedBy: This relation signifies the creation of an entity by an activity.
- used: This relation denotes that the functionality of an entity has been adopted by an activity.
- wasInformedBy: This relation denotes a causality that follows the exchange of two activities.

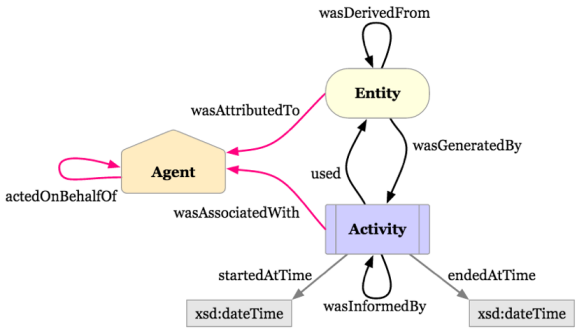

Fig. 3. Prov-DM respresentation showing various types contained in the model (Entity, Activity, and Agent)

- wasDerivedFrom: This relation represents a copy of information from an entity.
- wasAttributedTo: This denotes relational dependency to an agent. It is used to denote relationship between entity and agent when the activity that created the agent is unknown.
- wasAssociatedWith: This relation denotes a direct association to an agent for an activity that occurs. This indicates that an agent plays a role in the creation or modification of the activity.
- actedOnBehalfOf: This relation denotes assigning authority to perform a particular responsibility to an agent. This could be by itself or to another agent.

III. Related Works

There has been a considerable amount of work done on data provenance collection [**?**], [**?**], [**?**], [**?**]. Some of the work done has been focused on databases, sensor networks, scientific workflow systems and file system provenance but so far little attention has been given to provenance in IoT. Some of the prior work done on data provenance collection are outlined below:

A. Provenance Aware Storage System (PASS)

Muniswamy Reddy et al [**?**] developed a provenance collection system that tracks systemlevel provenance of the Linux file system. Provenance information is stored in the same location as the file system for easy accessibility, backup, restoration, and data management. Provenance information is collected and stored in the kernel space. PASS is composed of 3 major components: provenance collector, provenance storage,and provenance query. The collector keeps track of system level provenance. It intercepts system calls which are translated into provenance data and initially stored in memory as inode cache. Provenance data is then transferred to a file system in an kernel database, BerkleyDB. This database maps key value pairs for provenance data for fast index look up. PASS collects and stores provenance information containing a reference to the executable that created the provenance data, input files, a description of hardware in which provenance data is produced, OS information, process environment, process parameters, and other data such as a random number generator seed. PASS detects and eliminates cycles that might occur in provenance dependencies as a result of version avoidance. Cycles violate the dependency relationships between entities. For example, a child node could depend on a parent node and also be an ancestor of a parent node. PASS eliminates cycles by merging processes that might have led to the cycles. It also provides functionality for querying provenance data in the database. The query tool is built on top of BerkleyDB. For querying provenance, users can process queries using the provenance explorer. Query is done using commands such as MAKEFILE GENERATION which creates the sequence of events that led to the final state of a file or process. DUMP ALL, gets the provenance of the requested file.

Our approach looks at data provenance with data pruning as a key requirement since we are dealing with devices with limited memory and storage capabilities. We employ a policy based approach which allows provenance pruning by storing what provenance data is considered important to a specific organization. Also,unlike PASS, our system collects not just system level provenance but also application level provenance.

B. HiFi

Bates et al. [**?**] developed system level provenance collection framework for the Linux kernel using Linux Provenance Modules(LPM), this framework tracks system level provenance such as interprocess communication, networking, and kernel activities. This is achieved by mediating access to kernel objects using Linux Security Model(LSM) which is a framework that was designed for providing custom access control into the Linux kernel. It consists of a set of hooks which executed before access decision is made. LSM was designed to avoid problem created by direct system call interception. The provenance information collected from the kernel space is securely transmitted to the provenance recorder in the user space.

HiFi contains three components: provenance collector, provenance log and provenance handler. The collector and log are contained in the kernel space while the handler is contained in the user space. The log is a storage medium which transmits the provenance data to the user space. The collector uses LSM which resides in the kernel space. The collector records provenance data and writes it to the provenance log. The handler reads the provenance record from the log. This approach to collecting provenance data differs from our work since we focus on embedded systems and are concerned with input and output (I/O) data, which involves sensor and actuator readings. Additionally, HiFi deals with collecting system level events which might incur additional overhead when compared to collecting application level provenance. HiFi is engineered to work solely on the Linux operating system. Embedded systems that do not run on Linux OS will not be able to incorporate HiFi.

C. RecProv

RecProv [**?**] is a provenance system which records user-level provenance, avoiding the overhead incurred by kernel level provenance recording. It does not require changes to the kernel like most provenance monitoring systems. It uses Mozilla rr to perform deterministic record and replay by monitoring system calls and non deterministic input. Mozilla rr is a debugging tool for Linux browser. It is developed for the deterministic recording and replaying of the Firefox browser in Linux. Recprov uses PTRACE_PEEKDATA from ptrace to access the dereferenced address of the traced process from the registers. Mozilla rr relies on ptrace, which intercepts system calls to monitor the CPU state during and after a system call. It ptrace to access the dereferenced address of the traced process from the registers. System calls are monitored for file versioning. The provenance information generated is converted into PROV-JSON, and stored in Neo4j, a graph database for visualization and storage of provenance graphs.

D. Trustworthy Whole System Provenance for Linux Kernel

Bates et al [**?**] provide a security model for provenance collection in the linux kernel. This is achieved by creating a Linux Provenance Model (LPM). LPM serves as a security layer which provides a security abstraction for provenance data. It is involved in the attestation disclosure of the application layer and authentication channel for the network layer. The goal of LPM is to provide an end to end provenance capture system. LPM ensures the following security guarantees: For LPM,the system must be able to detect and avoid malicious forgery of provenance data. The system must be trusted and easily verifiable.

When an application invokes a system call, the LPM tracks system level provenance which is transmitted to the provenance module via a buffer. The provenance module registers contain hooks which records system call events. These events are sent to a provenance recorder in the user space. The provenance recorder ensures that the provenance data is stored in the appropriate backend of choice. Provenance recorders offer storage support for Gzip, PostGreSQL, Neo4j and SNAP.

E. StoryBook

Spillance et al [**?**] developed a user space provenance collection system, Storybook, which allows for the collection of provenance data from the user space thereby reducing performance overhead. This system takes a modular approach that allows the use of application specific extensions allowing additions such as database provenance, system provenance, and web and email servers. It achieves provenance capture intercepting system level events on FUSE, a file system and MySQL , a relational database. StoryBook allows developers to implement provenance inspectors these are custom provenance models which captures the provenance of specific applications which are often modified by different application(e.g web servers, databases). When an operation is performed on a data object, the appropriate provenance model is triggered and provenance data for that data object is captured. StoryBook stores provenance information such as open, close, read or write, application specific provenance, and causality relationship between entities contained in the provenance system. Provenance data is stored in key value pairs using Stasis and Berkely DB as the storage backend. It also allows the use of interoperable data specifications such as RDF to transfer data between various applications. Storybook allows for provenance query by looking up an inode in the ino hashtable. Collecting application level and kernel level events is similar to our approach of provenance collection, however, our approach integrates the use of a policy to eliminate noisy provenance data thereby allowing only relevant provenance to be stored.

F. Backtracking Intrusions

Samuel et al [**?**] developed a system, Backtracker, which generates an information flow of OS objects (e.g file, process) and events (e.g system call) in a computer system for the purpose of intrusion detection. From a detection point, information can be traced to pinpoint where a malicious attack occurred. The information flow represents a dependency graph which illustrates the relationship between system events. Detection point is a point in which an intrusion was discovered. Time is included in the dependency between objects to reduce false dependencies. Time is denoted by an increasing counter. The time is recorded as the difference of when a system call is invoked till when it ends. Backtracker is composed of two major components: EventLogger and GraphGen. An EventLogger is responsible for generating system level event log for applications running on the system. EventLogger is implemented in two ways: using a virtual machine and also as a stand alone system. In a virtual machine, the application and OS is run within a virtual machine. The OS running inside of the virtual machine is known as the guest OS while the OS on the bare-metal machine is known as the host OS, hypervisor or virtual machine monitor. The virtual machine alerts the EventLogger in the event that an application makes a system call and or exits. EventLogger gets event information, object identities, dependency relationship between events from the virtual machine's monitor and also the virtual machine's physical memory. EventLogger stores the collected information as a compressed file. EventLogger can also be implemented as a stand alone system which is incorporated in an operating system. Virtual machine is preferred to a standalone system because of the use of virtual machine allows ReVirt to be leveraged. ReVirt enables the replay of instruction by instruction execution of virtual machines. This allows for whole information capture of workloads. After the information has been captured by the EventLogger, GraphGen is used to produce visualizations that outlines the dependencies between events and objects contained in the system. GraphGen also allows for pruning of data using regular expressions which are used to filter the dependency graph and prioritize important portions of the dependency graph.

G. Provenance Recording for Services

Grouth et al [**?**] developed a provenance collection system, PReServ which allows software developers to integrate provenance recording into their applications. They introduce P-assertion recording protocol as a way of monitoring process documentation for Service Oriented Architecture (SOA) in the standard for grid systems. PReServ is the implementation of P-assertion recording protocol (PreP). PreP specifies how provenance can be recorded. PReServ contains a provenance store web service for recording and querying of P-assertions. P-assertions are provenance data generated by actors about the application execution. It contains information about the messages sent and received as well as state of the message. PReServ is composed of three layers, the message translator which is involved with converting all messages received via HTTP requests to XML format for the provenance store layer. The message translator also determines which plug-in handle to route incoming request to, the Plug-Ins layer. It implements functionality that the Provenance Store component provides. This component allows third party developers to add new functionality to store provenance data without going through details of the code implementation. The backend component stores the P-assertions. Various backend implementations could be attached to the system by the plug-in component.

PReServ was developed to address the needs of specific application domain (Scientific experiments). It deals with recording process documentation of Service Oriented Architecture (SOA) and collects P-assertions which are assertions made about the execution (i.e messages sent and received, state information) of actions by actors. The provenance collected by PReServ does not demonstrate causality and dependency between objects. In other words, the provenance data collected are not represented using a provenance model (e.g PROV-DM) that depicts causal relationship between provenance data.

H. Towards Automated Collection of Application-Level Data Provenance

Tariq et al. [**?**] all developed a provenance collection system which automatically collects interprocess provenance of applications source code at run time. This takes a different approach from system level provenance capture. It achieves provenance capture by using LLVM compiler framework and SPADE provenance management system. LLVM is a framework that allows dynamic compilation techniques of applications written in C, C++, Objective-C, and Java. Provenance data is collected from function entry and exit calls and is inserted during compilation. LLVM contains an LLVM reporter. This is a Java class that parses the output file collected from the LLVM reporter and forwards the provenance data collected to the SPADE tracer. SPADE is a provenance management tool that allows for the transformation of domain specific activity in provenance data. The LLVM Tracer module tracks provenance at the exit and entry of each function. The output is an Open Provenance Model in which functions are represented by an OPM process, arguments and return values are represented as an OPM artifact. To minimize provenance record, users are allowed to specify what functions they would like to collect provenance information at run time. The LLVM optimizer generates a graph of the target application. This graph is used to perform reverse reachability analysis from the functions contained in the graph. A workflow of how the framework works in collecting provenance is as follows:

- The application is compiled and converted into bitcode using the LLVM C compiler, clang.
- The LLVM Tracer module is compiled as a library.
- LLVM is run in combination with the Tracer, passed to include provenance data.
- Instrumentation bitcode is converted into assembly code via the compiler llc
- The socket code is compiled into assembly code.
- The instrumented application and assembly code is linked into an executable binary and sent to the SPADE kernel.

One major limitation to this approach of collecting application level provenance is that a user is required to have the source code of the specific application in which provenance data is required. Also, provenance collection is limited to function's exit and entry points. Provenance collection deals with data pruning by allowing a user to specify what provenance data to collect which is similar to our policy based approach for efficient provenance data storage.

I. Provenance from Log Files: a BigData Problem

Goshal et al [**?**] developed a framework for collecting provenance data from log files. They argue that log data contains vital information about a system and can be an important source of provenance information. Provenance is categorized based on two types: process provenance and data provenance. Process provenance involves collecting provenance information of the process in which provenance is captured. Data provenance on the other hand describes the history of data involved in the execution of a process. Provenance is collected by using a rule based approach which consists of a set of rules defined in XML. The framework consists of three major components. The rule engine which contains XML specifications for selecting, linking and remapping provenance objects from log files. The rule engine processes raw log files to structured provenance. There are three categories of rules as specified by the grammar. The match-set rules which selects provenance data based on a set of matching rules. Link rules which specifies relationship between provenance events and remap rules are used to specify an alias for a provenance object. The rule engine is integrated with the log processor. The log processor component is involved with parsing log files with information contained in the rule engine. It converts every matching log information to a provenance graph representing entities (vertices) and their relationship (edges). The adapter converts the structured provenance generated by the log processor into serialized XML format which is compatible with Karma, a provenance service application that is employed for the storage and querying of the provenance data collected.

J. Towards a Universal Data Provenance Framework using Dynamic Instrumentation

Gessiou et al [**?**] propose a dynamic instrumentation framework for data provenance collection that is universal and does not incur the overhead of kernel or source code modifications. This is achieved by using DTrace, a dynamic instrumentation utility that allows for the instrumentation of user-level and kernel-level code and with no overhead cost when disabled.

The goal is to provide an easy extension to add provenance collection on any application regardless of its size or complexity. Provenance collection is implemented on the file system using PASS and evaluated on a database(SQLite) and a web browser(Safari). The logging component monitors all system calls for processes involved. The logging component contains information such as system-call arguments, return value, user id, process name,and timestamp. The logging components includes functionalities to collect library and functional calls. The authors argue that applying data provenance to different layers of the software stack can provide varying levels of information. Provenance needs to be captured at the layer in which it is closest to the information that is needed to be captured. Provenance information that pertains to complex system activities such as a web browser or a database are collected. The information is collected from a process system-call and functional-call based on valid entry points contained in the process. DTrace dynamic instrumentation helps in discovering interesting paths in a system in which data propagates. This helps users use this information to collect a more accurate provenance data.

K. Provenance-Based Trustworthiness Assessment in Sensor Networks

Lim et al. [**?**] developed a model for calculating the trust of nodes contained in a sensor network by using data provenance and data similarity as deciding factors to calculate trust. The value of provenance signifies that the more similar a data value is, the higher the trust score. Also, the more the provenance of similar values differ, the higher their trust score. The trust score of a system is affected by the trust score of the sensor that forwards data to the system. Provenance is determined by the path in which data travels through the sensor network. This work differs from our approach since the authors focus on creating a trust score of nodes connected in a sensor network using data provenance and do not emphasize how the provenance data is collected. We are focused on creating a provenance aware system for sensor-actuator I/O operations which may be used to ensure trust of connected devices.

IV. Prov-Sensor Model Alignment

Sensor data contains observation data such as temperature, location information which could be represented in various standardized data interchange formats (RDF, XML, JSON e.t.c). This information is important but does not depict dependency relationships. PROV-O and sensor data can be aligned for better representation of dependencies between sensor data.

Using the PROV-O, In an IoT device connected to multiple sensors, the device and the sensors are represented as agents (prov:agents), the operation performed on the data (read, create, update) is represents an activity (prov:activity). The data being generated is known as an entity (prov:entity). Sensor readings generated from IoT devices are represented as a tuple $< t, d, s_1, r_1 >$ where t represents a timestamp, d represents data which is being collected, s_1 represents sensor information and r_1 represents device information. d can further be expanded as a list of data points. For example, consider sensor s_1, which generates sensor and humidity readings connected to device r_1, a raspberry pi, d can be defined as $d = [temperature, humidity]$ therefore the tuple representation of trace data for the raspberry pi r_1 with sensor s_1 is represented as $< t, [temperature, humidity], s_1, r_1 >$. Figure 4 further illustrates this concept with a graphical representation of the provenance-sensor alignment for a device connected to three sensors $s_1, s_2, and s_3$ with data $\{d_1, ..., d_4, e1, e2, f_1, ..., f_4\}$. Data is generated by three identical sensors, s_1, s_2 and s_3. All sensors generates data in real time. Trace generated from devices aligned with PROV-O which are represented as a graph. The graph depicts the dependency between the raspberry pi, r_1, the three sensors, the activity performed by the sensors (the sensors create data in this case) and the data generated. The dotted arrows represents time dependency between various entities.

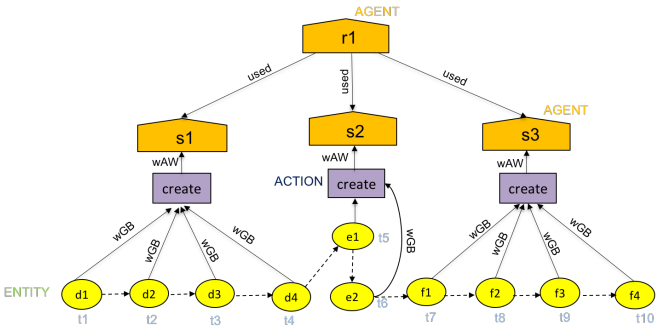

Fig. 4. Prov-Sensor Model Alignment

Provenance-Aware System Implementation

In this section, we outline components the provenance collection system and describe how provenance trace is collected across the IoT framework. Figure 6 displays the system architecture of our approach. Sensor and actuator readings in the form of input and output (I/O) events are recorded by the tracer component. This component intercepts system level I/O events and produces trace information represented in Common Trace Format (CTF). CTF encodes binary trace output information containing multiple streams of binary events such as I/O activity. Trace information is converted to provenance data in the PROV-DM IoT model and serialized to PROV-JSON. CTF conversion to PROV-DM will be achieved using babeltrace. This conversion can happen at any layer of the IoT stack.

Babeltrace is a plugin framework which allows the conversion of CTF traces into other formats. Trace or provenance data is securely transmitted to a gateway and later transmitted and stored in a cloud backend. Our backend of choice is Neo4j, a graph database for efficient storage, query and visualization of provenance data.

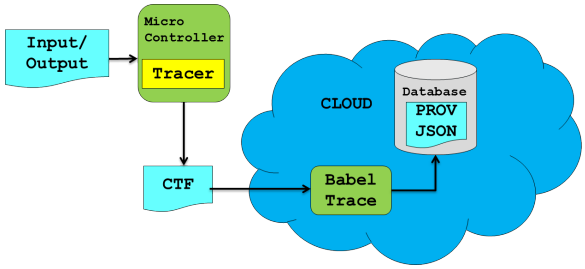

Fig. 5. System Architecture for Provenance Collection.

Our goal is to create a provenance-aware system which records I/O operations on data for devices connected in an IoT system. For our implementation, several tools and hardware components are utilized in the development of our prototype, outlined below:

- Raspberry Pi the microcontroller used to evaluate our approach. We choose Raspberry Pi because it is a representation of what can be found on an IoT gateway device and it has the capability to include custom hardware in programmable logic. Also, Raspberry Pi is a low cost, simple IoT demonstrator that was chosen for its high performance, onboard emulation, and IoT gateway projects can be programmed without additional need for hardware tools.
- Neo4j, a graph database which allows optimized querying of graph data. Since provenance represents causal dependencies, it is ideal to use a graph database to store the relationships between objects
- Babeltrace: This is a trace converter tool. It contains plugins used to convert traces from one format into another.
- barectf: This is an application that collects bare metal application trace in CTF.
- yaml generator: yaml generator creates yaml files. A yaml configuration file contains information on what barectf application needs in order to generate CTF trace output. This consist of configuration settings such as an application trace stream, packet type, payload type and size.

A. *Prov-Aware IoT Information Flow*

Place holder...

V. Conclusion

In this paper, we motivate the need for integrating provenance into the IoT architecture. We propose a provenance collection framework that provides provenance collection capabilities for devices in the IoT.

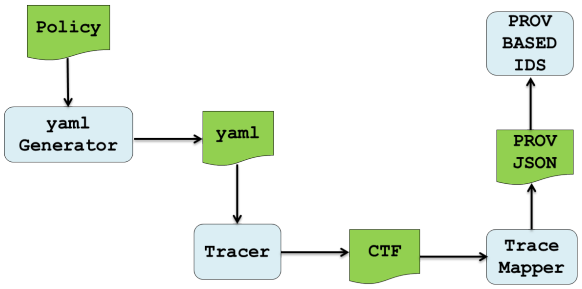

Fig. 6. System Architecture for Provenance Collection.

VI. Future Work

Some of the proposed future work for this research is:

- Provenance collection raises privacy issues. How do we ensure that the vast amount of data collected is not invasive to privacy.
- Securing Provenance: Proper encryption and authentication techniques [**?**] are needed to ensure the confidentiality, and integrity of provenance data.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.